Session 408

OpenGL: Advanced Optimizations

John Stauffer
OpenGL Engineering Manager

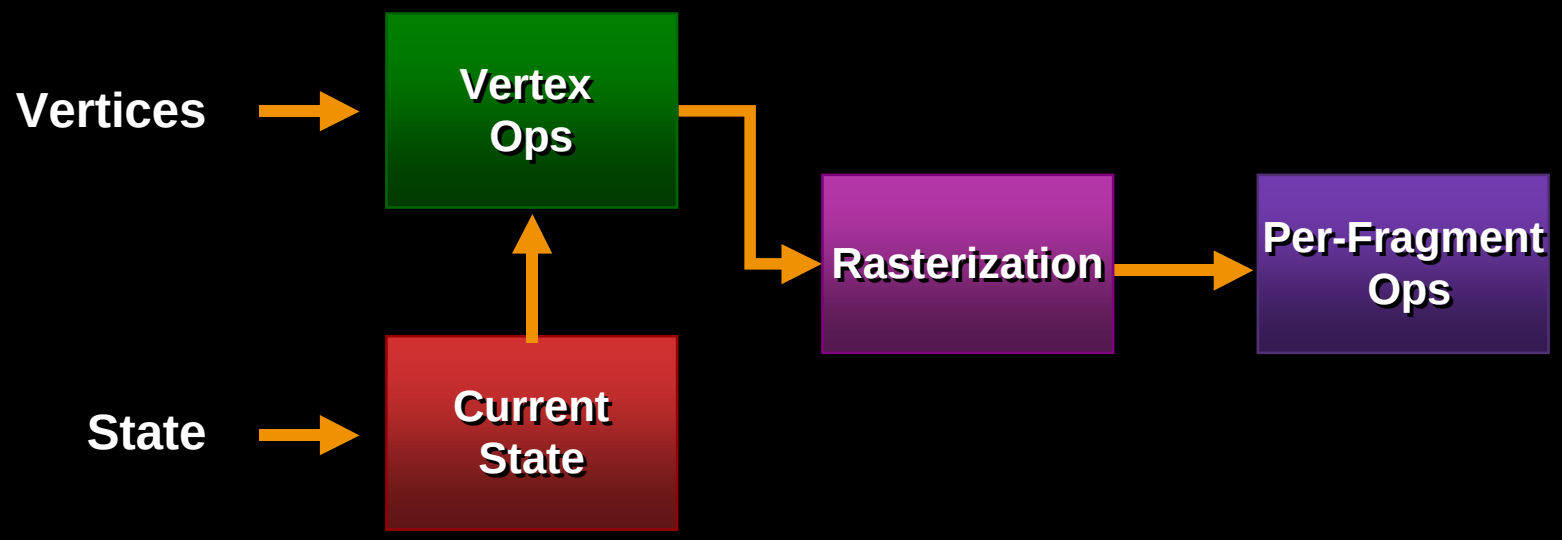
What You'll Learn

- Key areas for optimizing OpenGL
  - Application
  - Setup
  - State management
  - Texture management
  - Vertex operations
  - Per-fragment operations

What You'll Learn

- Extensions that help optimize OpenGL
- Multi-CPU utilization
- Where to look for help

OpenGL Pipeline



Application

- What should the minimum performance for your application be?
  - Is user input granularity important
  - How critical is display quality
- What is the target platform?
  - Minimum VRAM required
  - Minimum memory required
  - OpenGL extensions required
- Allow the user to choose between quality and performance

Application

- Run without OpenGL rendering enabled to get the application performance
  - Pass a NULL as the current OpenGL context

```
CGLSetCurrentContext(NULL);
                Or
[NSOpenGLContext clearCurrentContext];
```

  - Measure the time spent in the application

```
gettimeofday(&t1, NULL);
***  Application code ***
gettimeofday(&t2, NULL);
app_time = 1000000.0f*(t2.tv_sec-t1.tv_sec)+(t2.tv_usec-t1.tv_usec);
```

  - Calculate the open loop FPS

```
app_fps = 1.0f / app_time;
```

Application

- Estimating actual performance

```
estimated_fps = app_fps * 0.75f;
```

- Optimizing application code
  - Use Sampler to profile your application
    - Works to find hot spots
    - Sampler works with Macho and CFM apps
    - /Developer/Applications/Sampler

OpenGL Setup

- Select the best display(s)
  - Getting the main display

```
display = kCGDirectMainDisplay;
display_mask = CGDisplayIDToOpenGLDisplayMask(display);
```

  - Getting all active displays

```
CGGetActiveDisplayList(8, &activeDspys, &dspyCnt);

for(i = 0; i < dspyCnt; i++)
{
    display_mask  |=
    CGDisplayIDToOpenGLDisplayMask(activeDspys[i]);
}
```

OpenGL Setup

- Find VRAM size

```
CGLQueryRendererInfo(display_mask, rend, &nrend);

for(i = 0; i < nrend; i++)
{
    CGLDescribeRenderer(rend, i, kCGLRPAccelerated, &accel);
   if(accel)
   {
        CGLDescribeRenderer(rend, i, kCGLRPVideoMemory,
           &vram);
        if(vram < min_vram )  min_vram   = value;
        if(vram > max_vram ) max_vram  = value;
   }
}

CGLDestroyRendererInfo(rend);
```

OpenGL Setup

- Setup OpenGL based on VRAM size
  - Texture sizes
  - Screen resolution
  - Pixel depth
- Setup OpenGL based on CPU type
  - Default to simpler data set for older CPUs

State Management

- Minimize state changes
  - State changes can account for a large portion on the time in OpenGL
- Avoid calling glFlush
  - Submits the current command buffer
  - Can artificially cause the driver to run out of command buffers
- Avoid calling glFinish
  - Waits until all pending drawing is finished by the graphics device

State Management

- Avoid reading data back
  - Textures, State, Frame buffer
  - Can cause an implicit glFinish
- Avoid overly complex state settings
  - Can vary from card to card
  - Lighting, User clipping planes, AA

Texture Management

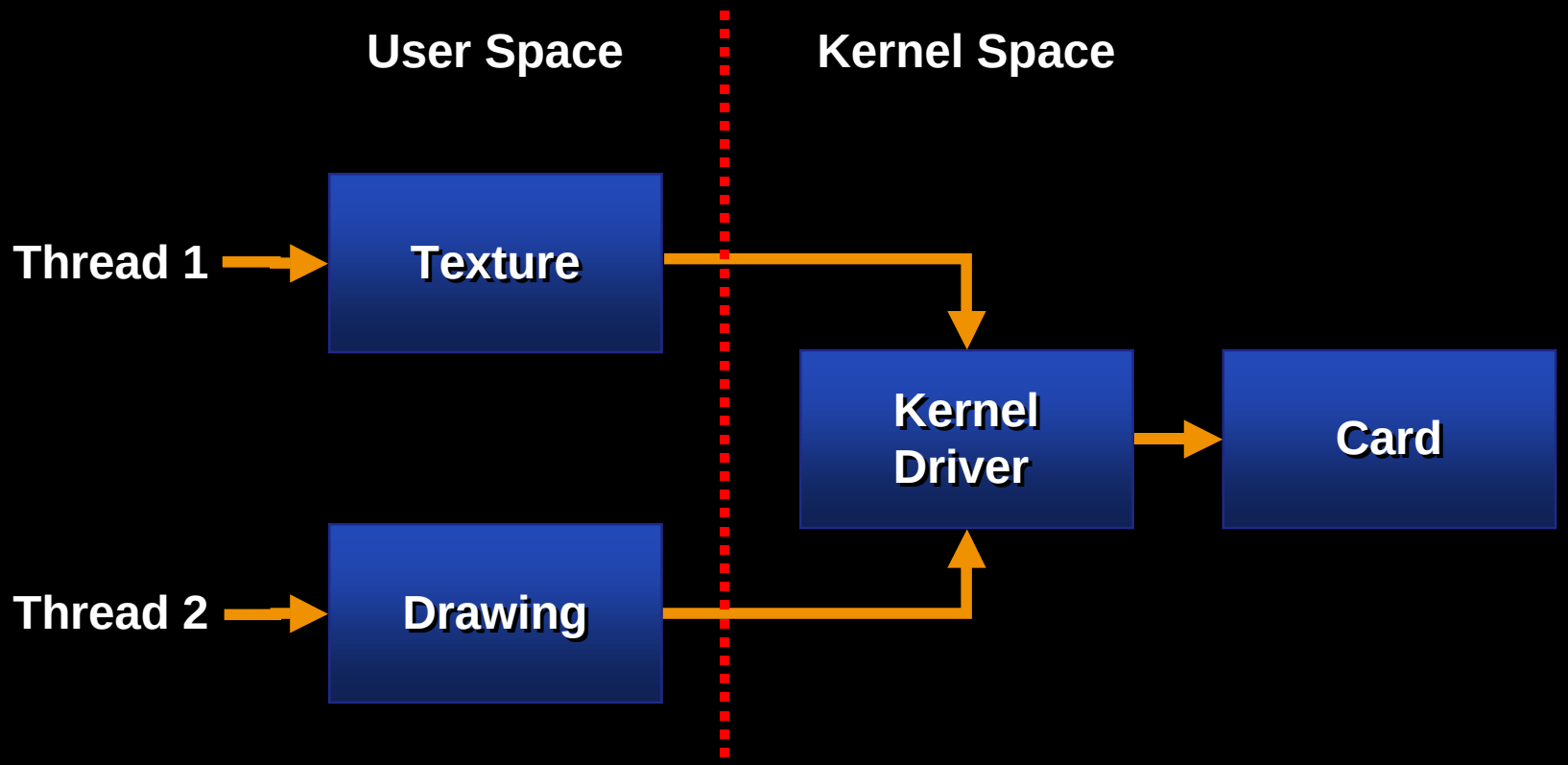
- Avoid uploading a texture more than once
- Avoid keeping a copy of the texture
  - Saves memory
- Use a packed pixel format
  - GL_APPLE_packed_pixel
  - Offers the fastest texture upload
  - Not all formats are supported natively
- Minimize changing the current texture
  - One of the most expensive state changes

Texture Management

- Scale texture sizes to hardware
  - Try to avoid texture paging
- Don’t second guess OpenGL texture paging
  - OpenGL uses LRU/MRU paging
  - Paging uses very little CPU
- Split large textures in tiles
  - Interlacing texture uploads with drawing

Texture Management

- Load textures on a separate thread



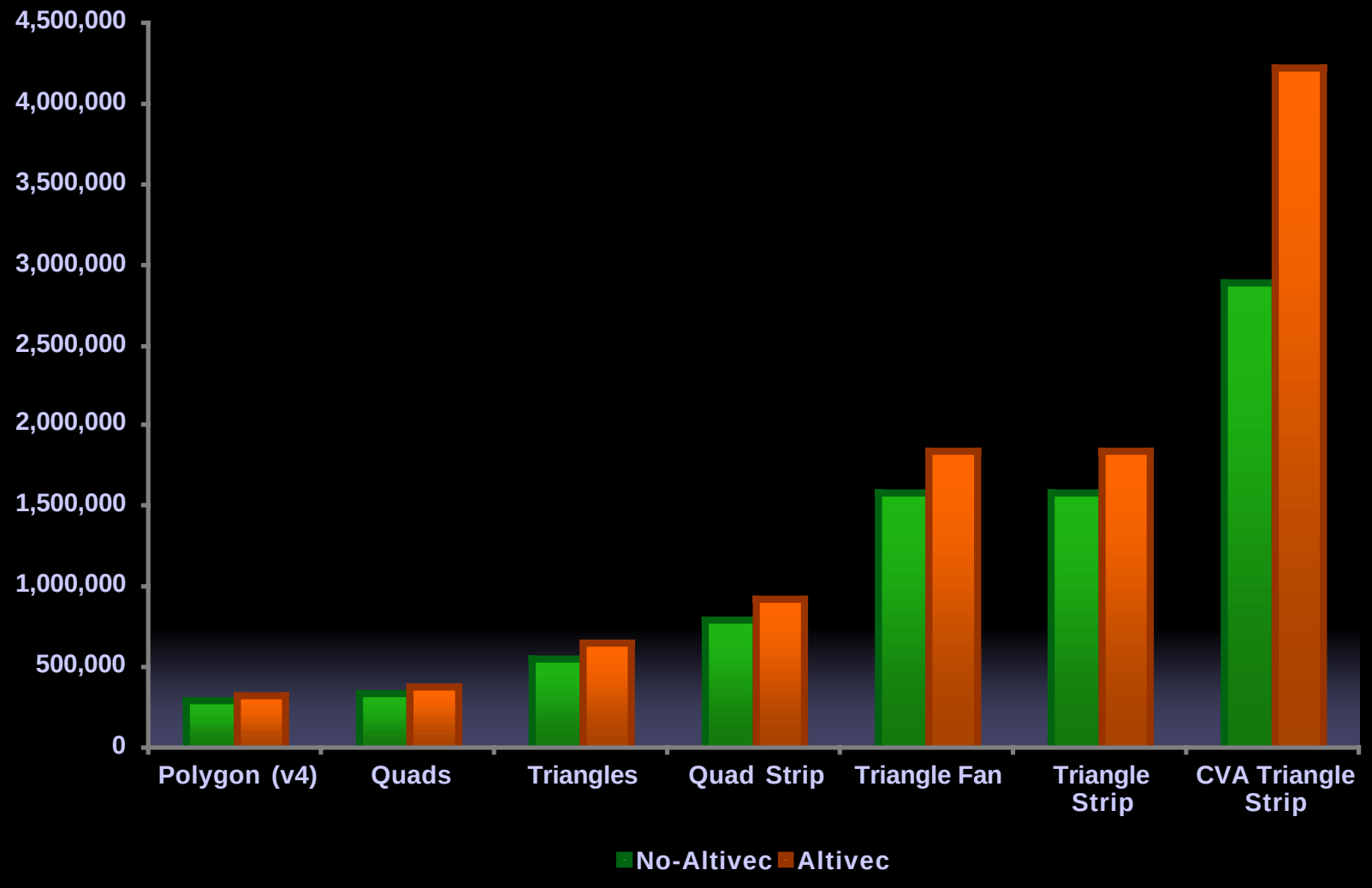
Demo

John Stauffer
3D Engineering Manager

Vertex Operations

- Maximize number of vertices per glBegin/glEnd pair
  - This reduces per function call overhead
- Use efficient primitives
  - Triangle strips
- Use the vertex arrays where possible
- Use GL_EXT_compiled_vertex_array
  - Use for large vertex arrays (>16)

Two Texture Performance



Optimization Example

- Step 1: Starting point

```
for(i = 0; i < num_polys - 2; i++)
{
        glShadeModel(GL_SMOOTH);
        glColor4d (r, g, b, 1.0);
        glBegin (GL_TRIANGLES);
                glVertex4f (v[i][0], v[i][1], v[i][2], 1.0);
                glVertex4f (v[i+1][0], v[i+1][1], v[i+1][2],
1.0);
                glVertex4f (v[i+2][0], v[i+2][1], v[i+2][2],
1.0);
        glEnd ();
}
```

Optimization Example

- Step 2: Remove static state changes from loops

```
glShadeModel(GL_SMOOTH);
glColor4d (r, g, b, 1.0);
for(i = 0; i < num_polys - 2; i++)
{
        glBegin (GL_TRIANGLES);
                glVertex4f (v[i][0], v[i][1], v[i][2], 1.0);
                glVertex4f (v[i+1][0], v[i+1][1], v[i+1][2], 1.0);
                glVertex4f (v[i+2][0], v[i+2][1], v[i+2][2], 1.0);
        glEnd ();
}
```

Optimization Example

- Step 3: Keep state as simple as possible

```
glShadeModel(GL_FLAT);
glColor4d (r, g, b, 1.0);
for(i = 0; i < num_polys - 2; i++)
{
        glBegin (GL_TRIANGLES);
                glVertex4f (v[i][0], v[i][1], v[i][2], 1.0);
                glVertex4f (v[i+1][0], v[i+1][1], v[i+1][2],
1.0);
                glVertex4f (v[i+2][0], v[i+2][1], v[i+2][2],
1.0);
        glEnd ();
}
```

Optimization Example

- Step 4: Maximize vertices per begin/end

```
glShadeModel(GL_FLAT);
glColor4d (r, g, b, 1.0);
glBegin (GL_TRIANGLES);
        for(i = 0; i < num_polys - 2; i++)
        {
                glVertex4f (v[i][0], v[i][1], v[i][2], 1.0);
                glVertex4f (v[i+1][0], v[i+1][1], v[i+1][2],
1.0);
                glVertex4f (v[i+2][0], v[i+2][1], v[i+2][2],
1.0);
        }
glEnd ();
```

Optimization Example

- Step 5: Simplify data types

```
glShadeModel(GL_FLAT);
glColor3f (r, g, b);
glBegin (GL_TRIANGLES);
        for(i = 0; i < num_polys - 2; i++)
        {
                glVertex3fv (v[i]);
                glVertex3fv (v[i+1]);
                glVertex3fv (v[i+2]);
        }
glEnd ();
```

Optimization Example

- Step 6: Simplify primitive types

```
glShadeModel(GL_FLAT);
glColor3f (r, g, b);
glBegin (GL_TRIANGLE_STRIP);
        for(i = 0; i < num_polys; i++)
        {
                glVertex3fv (v[i]);
        }
glEnd ();
```

Optimization Example

- Step 7: Further simplify primitive types

```
glShadeModel(GL_FLAT);
glColor3f (r, g, b);

glVertexPointer (3, GL_FLOAT, 0, v);
glEnableClientState (GL_VERTEX_ARRAY);
glDrawElements (GL_TRIANGLE_STRIP, num_polys,
                GL_UNSIGNED_SHORT, indices);
```

Demo

John Stauffer
3D Engineering Manager

Per-Fragment Ops

- Use multitexture instead of multipass rendering
  - Reduces the CPU burden
  - Reduces frame buffer traffic
- Avoid read/modify/write operations on the frame buffer
  - Blending, Z-buffer test
  - Reduces frame buffer traffic

Per-Fragment Ops

- Render from front to back
  - Allows hardware hierarchical Z to be effective
  - Reduces frame buffer traffic

OpenGL Extensions

- GL_EXT_compiled_vertex_array
  - Good for drawing more 16 vertices at a time
  - Reduces number of transformations
  - Allows vertices to be precompiled into hardware buffers
- GL_ARB_texture_compression
  - Saves system/VRAM memory
  - Reduces frame buffer traffic

OpenGL Extensions

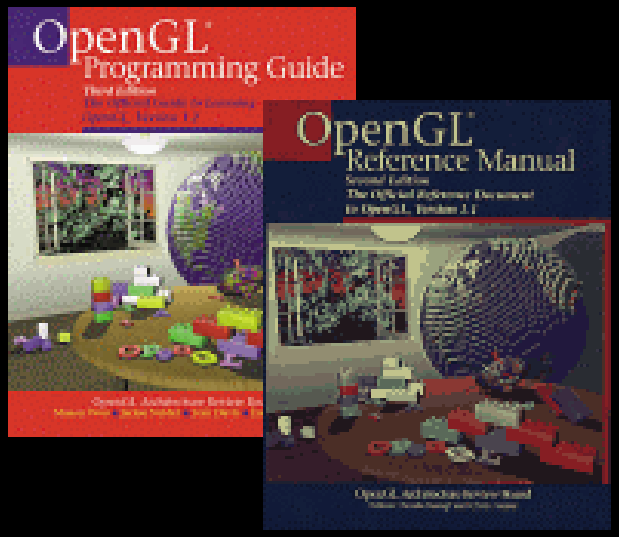
- GL_ARB_multitexture
  - Apply multiple textures in a single pass
- GL_APPLE_packed_pixel
  - Optimizes texture and bitmap images loading
  - Saves system memory

Summary

- Optimize your application
- An application should scale to the platform
  - VRAM
  - System memory
  - Number texture units
  - CPU type
  - OpenGL extensions
- Allow user to adjust the graphical quality

For More Information

- In print
  - *OpenGL Programming Guide*
  - *OpenGL Reference Manual*



- On-line
  - **<http://www.opengl.org>**
  - **<http://lists.apple.com>**
    - Search for “opengl”

Who to Contact

Sergio Mello
3D Developer Support
Apple Worldwide Developer Relations
sergio@apple.com

Roadmap

FF110 Feedback Forum for OpenGL	Room J1 **Thurs., 5:00pm**
409 OpenGL: Advance Rendering	Room J2 **Thurs., 10:30am**

Q&A

http://developer.apple.com/events/urls.html

WWDC

Worldwide Developers Conference 2001